The Lifted Matrix-Space Model for Semantic Composition

WooJin Chung[1]
woojin@nyu.edu

Samuel R. Bowman[1,2]
bowman@nyu.edu

[1]Dept. of Linguistics
New York University
10 Washington Place
New York, NY 10003

[2]Center for Data Science
New York University
60 Fifth Avenue
New York, NY 10011

Abstract

Recent advances in tree structured sentence encoding models have shown that explicitly modeling syntax can help handle compositionality. More specifically, recent works by Socher et al. (2012), Socher et al. (2013), and Chen et al. (2013) have shown that using more powerful composition functions with multiplicative interactions within tree-structured models can yield significant improvements in model performance. However, existing compositional approaches which make use of these multiplicative interactions usually have to learn task-specific matrix-shaped word embeddings or rely on third-order tensors, which can be very costly. This paper introduces the Lifted Matrix-Space model which improves on the predecessors on this aspect. The model learns a global transformation from pre-trained word embeddings into matrices, which can be composed via matrix multiplication. The upshot is that we can capture the multiplicative interaction without learning matrix-valued word representations from scratch. In addition, our composition function effectively transmits a larger number of activations across layers with comparably few model parameters. We evaluate our model on the Stanford NLI corpus and the Multi-Genre NLI corpus and find that the Lifted Matrix-Space model outperforms the tree-structured long short-term memory networks.

1 Introduction

Encoding sentence meanings based on parse trees has been a popular line of research. Modern theories in natural language syntax suggest that sentences are tree-structured, and the meaning of each node is calculated from the meaning of its child nodes. *Semantic composition* refers to the process of composing the child node representations, and the meaning of a whole sentence is constructed by recursively applying this methodology from the bottom up. From a functional point of view, the parent node is a function of its child nodes, thus semantic composition can be characterized in terms of *composition function*. The success of a tree-structured model largely depends on the design of the composition function.

It has been empirically attested that a composition function that captures multiplicative interaction yields better results (Rudolph and Giesbrecht 2010; Socher et al. 2012; Socher et al. 2013). This paper presents a novel model which advances this line of research, the Lifted Matrix-Space model. We introduce a tensor-based LIFT layer that learns to produce matrix representations of words that are dependent on the content of pre-trained word embedding vectors. Composition of two matrix representations is carried out by a composition layer, into which the two matrices are sequentially fed. Figure 1 illustrates the model design.

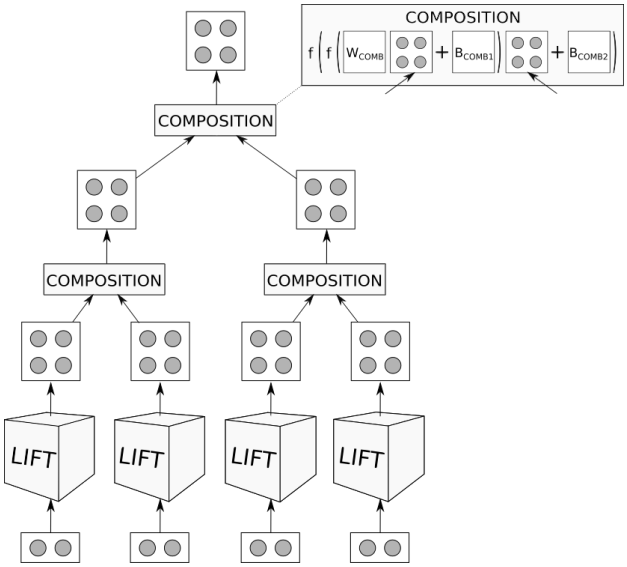


Figure 1: Lifted Matrix-Space model. The gray boxes denote composition layers and the gray cubes deonote lift layers.

Our model was inspired by Continuation Semantics (Barker and Shan 2014), where each symbolic representation of words is converted to a higher-order function. There is a consensus in linguistic semantics that a subset of natural language expressions correspond to higher-order functions. Inspired by the works in programming language theory, Continuation Semantics takes a step further and claims that all expressions have to be converted into a higher-order function before they participate in semantic composition. The theory bridges a gap between linguistic semantics and prgramming language theory, and reinterprets various linguistic phenomena such as pronoun resolution from the view

of computation. While we do not directly implement Continuation Semantics, we convert low-level representations (vectors) to higher-order functions (matrices), and composition only takes place between the higher-order functions.

A number of models have been developed to capture multiplicative interaction between distributed representations, including the Matrix-Vector recursive neural networks (Matrix-Vector RNN; Socher et al. 2012) and the Recursive Neural Tensor Network (Socher et al. 2013). While having a similar objective, the proposed model requires fewer model parameters than the predecessors because it does not learn each word matrix representation separately, and the number of parameters for the composition function is not proportional to the cube of the hidden state dimension. Thus, our model is more scalable than its predecessors and can be trained on substantially larger corpora.

We evaluate our model on the task of *natural language inference* (NLI). The task is to determine the inferential relation between a given pair of sentences, that is, whether the two sentences are in an *entailment*, *contradiction*, or *neutral* relation. It is one of the most principled test cases among natural language processing tasks, because knowing the inferential relations is closely related to understanding the meaning of an expression (Chierchia and McConnell-Ginet 1990); one way of characterizing the meaning of a sentence is to identify the set of sentences that it entails. While other tasks such as question answering or machine translation require a model to learn task-specific behavior that goes beyond understanding sentence meaning, the focus of NLI remains narrow, requiring less engineering efforts.

2 Related works

This section presents extant neural network models for semantic composition, primarily focusing on the ones that take advantage of tree structures. While the tree structure-based models are not the only game in town, they are well-suited for applying the principle of compositionality (Frege 1892, Dowty 2007); tree structure-based models develop a composition function which computes the meaning of each node based on the meaning of its child nodes. The task of modeling semantic composition has been thoroughly studied in recent years (Mitchell and Lapata 2010, Baroni and Zamparelli 2010, Zanzotto et al. 2010, Socher et al. 2011). Although this line of approach has been very successful, many of the extant models ultimately rely on element-wise addition of vectors. The Tree-structured Recursive Neural Network (TreeRNN) of Socher, Manning, and Ng (2010) is a representative case, where two child node vectors $\vec{h}_{left}$ and $\vec{h}_{right}$ are composed via the following function:

$$(1) \quad \vec{h} = tanh(W \begin{bmatrix} \vec{h}_{left} \\ \vec{h}_{right} \end{bmatrix} + \vec{b})$$

where $\vec{h}_{left}$, $\vec{h}_{right}$, $\vec{h}$, $\vec{b} \in \mathbb{R}^{d \times 1}$, and $W \in \mathbb{R}^{d \times 2d}$. Here and throughout, d stands for the number of activations. The equation in (1) can be rewritten as in (2), where $W = [W_{left}; W_{right}]$. In other words, the composition function merely multiplies a weight matrix to each child vector, calculates the element-wise addition of the two and a bias, and feeds the resulting column vector to a non-linear activation function.

$$(2) \quad \vec{h} = tanh(W_{left}\vec{h}_{left} + W_{right}\vec{h}_{right} + \vec{b})$$

However, there is no reason to believe that element-wise addition is adequate for modeling semantic composition. Formal work in linguistic semantics has shown that many linguistic expressions are well-represented as functions. Accordingly, composing two meanings typically require feeding an argument into a function (function application). Such an operation involves a complex interaction between the two meanings, but the classic TreeRNN does not supply any additional means to capture the interaction.

Another issue is that the majority of the models rely on pre-trained word embeddings. The vast majority of the pre-trained word embeddings are built based on the distributional hypothesis. In other words, the meaning of a word is determined by the words it co-occurs with in a given corpus. While this line of approach is useful in capturing syntactic aspects of words and has been successful in recognizing word similarity (Collobert et al. 2011) and recovering analogy (Mikolov et al. 2013), it is not ideal for understanding operator semantics. For instance, the operator meanings of quantifiers such as *every*, *no*, and *some* seem to be rather difficult to capture. This is because their distribution is not suggestive of how each of them contributes to the meaning of the sentence. As a representative example, *every* and *no* are quantifiers that are likely to co-occur with similar words. Even though the two quantifiers have crucially different meanings, any pre-trained word embeddings based on the distributional hypothesis would assign similar vectors to the two words.[1]

Rudolph and Giesbrecht's (2010) Compositional Matrix-Space model represents words and phrases as matrices and uses matrix multiplication to perform semantic composition.

$$(3) \quad P = AB$$

where A, B, $P \in \mathbb{R}^{d \times d}$ are matrix representations of the word embeddings. They provide a formal proof that element-wise addition/multiplication of vectors can be subsumed under matrix multiplication. Moreover, they claim that the order-sensitivity of matrix multiplication is adequate for capturing the semantic composition because natural language is order-sensitive. Yessenalina and Cardie (2011) utilize the Compositional Matrix-Space model and successfully applies it to sentiment analysis.

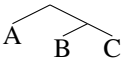
As Socher et al. (2012) point out, the problem of the Compositional Matrix-Space model is that the compositional aspect of meaning is lost due to the associative character of matrix multiplication. For instance, the following two tree

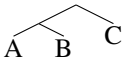
[1] There are two paths we can take to alleviate this issue: we can improve the quality of word embeddings (Hill et al. 2017), or add an extra neural network layer to help tearing apart such words (Irsoy and Cardie 2014). The two solutions are not mutually exclusive and thus can compensate each other. While we acknowledge the importance of quality word embeddings, we take on the latter path.

Model	Params.	Compositional	Multiplicative interaction
TreeRNN (Socher, Manning, and Ng 2010)	O($d \times d$)	Yes	No
TreeLSTM (Tai, Socher, and Manning 2015)	O($d \times d$)	Yes	No
Compositional Matrix-Space (Rudolph and Giesbrecht 2010)	O($V \times d \times d$)	No	Yes
Matrix-Vector RNN (Socher et al. 2012)	O($V \times d \times d$)	Yes	Yes
RNTN (Chen et al. 2013, Socher et al. 2013)	O($d \times d \times d$)	Yes	Yes
Lifted Matrix-Space (our model)	O($d \times d_{emb}$)	Yes	Yes

Table 1: Summary of the models. *Params.* is the approximate number of model parameters, d is the number of activations, d_{emb} is the dimension of word embeddings, and V is the size of the vocabulary.

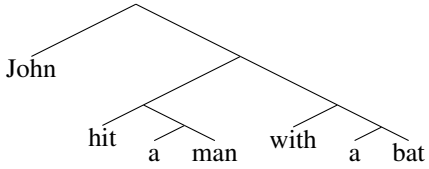
structures in (4) are syntactically distinct, but the Compositional Matrix-Space model would produce the same result for both structures because its mode of composition is associative.

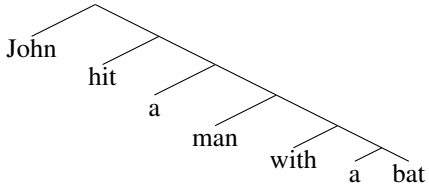
(4) a. 

b. 

The Compositional Matrix-Space model cannot distinguish the meaning of the two tree structures and invariably produces ABC (a sequence of matrix multiplications). Therefore, the information on syntactic constituency would be lost. This makes the model less desirable for handling semantic composition of natural language expressions for two reasons: First, the principle of compositionality is violated. Much of the success of the tree-structured models can be credited to the shared hypothesis that the meaning of every tree node is derived from the meanings of its child nodes. Abandoning this principle of compositionality gives up the advantage. Second, it cannot handle structural ambiguities exemplified in (5).

(5) John hit a man with a bat.

(6) a. 

b. 

The sentence has two interpretations that can be disambiguated with the following paraphrases: (i) John hit a man by swinging a bat, and (ii) John hit a man who is holding a bat. The common syntactic analysis of the ambiguity is that the prepositional phrase *with a bat* attaches to two different locations. If it attaches to the verb phrase *hit a man*, the first interpretation arises. On the other hand, if it attaches to the noun *man*, the second interpretation is given. However, if the structural information is lost, we would have no way to disambiguate the two readings.

Socher et al.'s (2012) Matrix-Vector RNN aims to capture the multiplicative interaction between two vectors, while conforming to the principle of compositionality. They hypothesize that representing operators as matrices can better reflect operator semantics. For each lexical item, a matrix is assigned in addition to the pre-trained word embedding vector. The matrices are part of the trained parameters, and the model aims to assign the right matrix representations to operators while assigning an identity matrix to words with no operator meaning. One step of semantic composition is defined as follows:

$$(7) \quad \vec{h} = f_{A,B}(\vec{a}, \vec{b}) = f(B\vec{a}, A\vec{b}) = g(W \begin{bmatrix} B\vec{a} \\ A\vec{b} \end{bmatrix})$$

$$(8) \quad H = f_M(A, B) = W_M \begin{bmatrix} A \\ B \end{bmatrix}$$

where $\vec{a}$, $\vec{b}$, $\vec{h} \in \mathbb{R}^{d \times 1}$, A, B, $H \in \mathbb{R}^{d \times d}$, and W, $W_M \in \mathbb{R}^{d \times 2d}$.

The Matrix-Vector RNN falls short on two aspects. First, there is an additional matrix to be learned for each of the lexical items. In addition to the large number of parameters required by the TreeRNN, $V \times d \times d$ parameters are added, where V is the size of the vocabulary and d is the number of activations. It is empirically known that the size of the vocabulary is roughly proportional to the size of the corpus (Heaps' law), therefore the number of model parameters increases as the corpus gets bigger. This makes the model less ideal for handling a large corpus: having a huge number of parameters causes a problem both for memory usage and for learning efficiency.

Second, the composition of the matrices that represent operator semantics does not capture multiplicative interaction. In (8), the matrix of the parent node, H, is produced by concatenating the two child matrices and multiplying the result with a weight matrix. This means that the composition of operators can be reduced to element-wise addition as in the case of the TreeRNN. It would be preferable to employ a more powerful composition function for the operators as well, because many natural language expressions require stacking of operators. For example, constituent negation can apply to a quantifier, which has its own complex operator semantics, to produce a new operator as in *not every*.

Chen et al. (2013) and Socher et al. (2013) present the Recursive Neural Tensor Network (RNTN) which reduces the

computational complexity of the Matrix-Vector RNN, while capturing the multiplicative interaction between child vectors. The model introduces a third-order tensor $\boldsymbol{V}$ which interacts with the child node vectors as follows:

$$\vec{h} = tanh(W \begin{bmatrix} \vec{h}_{left} \\ \vec{h}_{right} \end{bmatrix} + \vec{b} + \vec{h}_{left}^{T} \mathbf{V} \vec{h}_{right}) \quad (9)$$

where $\vec{h}_{left}$, $\vec{h}_{right}$, $\vec{b} \in \mathbb{R}^{d \times 1}$, $W \in \mathbb{R}^{d \times 2d}$, and $V \in \mathbb{R}^{d \times d \times d}$. The RNTN improves on the Matrix-Vector RNN in that its parameter size is not proportional to the size of the corpus. However, the addition of the third-order tensor V of dimension $d \times d \times d$ still requires proportionally more parameters.

The last composition function relevant to this paper is the Tree-structured long short-term memory (TreeLSTM) networks (Tai, Socher, and Manning 2015, Zhu, Sobhani, and Guo 2015, Le and Zuidema 2015), particularly the version over a constituency tree. It is an extension of the TreeRNN which adapts long short-term memory (Hochreiter and Schmidhuber 1997) networks. It shares the advantage of LSTM networks in that it prevents the vanishing gradient problem (Hochreiter et al. 2001).

Unlike the TreeRNN, the output hidden state $\vec{h}$ of the TreeLSTM is not directly calculated from the hidden states of its child nodes, $\vec{h}_{left}$ and $\vec{h}_{right}$. Rather, each node in the TreeLSTM maintains a cell state $\vec{c}$ that keeps track of important information of its child nodes. The output hidden state $\vec{h}$ is drawn from the cell state $\vec{c}$ by passing it through an output gate $\vec{o}$.

The cell state is calculated in three steps: (i) The TreeLSTM first computes a new candidate $\vec{g}$ from $\vec{h}_{left}$ and $\vec{h}_{right}$. It selects which values to take from the new candidate $\vec{g}$ by passing it through an input gate $\vec{i}$. (ii) The TreeLSTM then chooses which values to forget from the cell states of the child nodes, $\vec{c}_{left}$ and $\vec{c}_{right}$. For each child node, an element-wise product ($\odot$) between its cell state and the forget gate (either $\vec{f}_l$ and $\vec{f}_r$, depending on the child node) is calculated. (iii) Lastly, the results from (i) and (ii) are summed up.

$$\begin{bmatrix} \vec{i} \\ \vec{f}_l \\ \vec{f}_r \\ \vec{o} \\ \vec{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} \left(W \begin{bmatrix} \vec{h}_{left} \\ \vec{h}_{right} \end{bmatrix} + \vec{b} \right) \quad (10)$$

$$\vec{c} = \vec{f}_l \odot \vec{c}_{left} + \vec{f}_r \odot \vec{c}_{right} + \vec{i} \odot \vec{g} \quad (11)$$

$$\vec{h} = \vec{o} \odot \vec{c} \quad (12)$$

To summarize, the TreeRNN and the TreeLSTM reflect the principle of compositionality but cannot capture the multiplicative interaction between two expressions. In contrast, the Compositional Matrix-Space model incorporates multiplicative interaction but violates the principle of compositionality. The Matrix-Vector RNN is compositional and also captures the multiplicative interaction, but the number of parameters is roughly proportional to the size of the vocabulary. This makes the model inadequate for larger datasets. The RNTN is also compositional and integrates multiplicative interaction, but it requires less parameters than the Matrix-Vector RNN. In addition, a number of tests indicate that it shows better performance than aforementioned competitors. Nevertheless, it requires significantly more parameters than the TreeRNN or the TreeLSTM.

Just like the Matrix-Vector RNN and the RNTN, we aim to design a model that captures the multiplicative interaction between distributed representations. At the same time, we improve on the predecessors in terms of scalability, making the model more suitable for larger datasets.

3 Introducing the Lifted Matrix-Space model

We present the Lifted Matrix-Space model which renders semantic composition in a novel way. Despite the aforementioned difficulties that the Compositional Matrix-Space model has, we believe that matrix multiplication better captures the notion of semantic composition than element-wise addition or weighted average. As Socher, Manning, and Ng (2010) noted, the former is capable of modeling operator meanings which is essential to understanding natural language semantics. For this reason, our model inherits the main insights of the Compositional Matrix-Space model and extends it.

3.1 Base model

Our model primarily consists of two layers: the LIFT layer and the composition layer. The LIFT layer takes a pre-trained word embedding (represented as a vector) and outputs a corresponding matrix. Each word is transformed into a $\sqrt{d} \times \sqrt{d}$ matrix. The LIFT layer is defined as follows:

$$H = tanh(\boldsymbol{W}_{\mathbf{LIFT}} \vec{c} + B_{\text{LIFT}}) \quad (13)$$

where $\vec{c} \in \mathbb{R}^{d_{emb} \times 1}$, $B_{\text{LIFT}} \in \mathbb{R}^{\sqrt{d} \times \sqrt{d}}$, and $\boldsymbol{W}_{\mathbf{LIFT}} \in \mathbb{R}^{\sqrt{d} \times \sqrt{d} \times d_{emb}}$. The resulting H is an $\sqrt{d} \times \sqrt{d}$ matrix representation for each lexical item. It serves as the input for the composition layer.

The composition layer can be divided into two sublayers. Given two child node representations H_{c_left} and H_{c_right}, the first sublayer takes the matrix H_{c_left} and returns a hidden state $H_{inner} \in \mathbb{R}^{\sqrt{d} \times \sqrt{d}}$ as shown in (14). Since this hidden state is also a matrix, it can function as the weight matrix for the second sublayer. The second sublayer multiplies the hidden state H_{inner} with the matrix H_{c_right}, adds a bias, and feeds the result into a non-linear activation function, as in (15). The output of the second sublayer is the output of the entire composition function.

$$H_{inner} = tanh(W_{\text{COMB}} H_{c\_left} + B_{\text{COMB1}}) \quad (14)$$

$$H = tanh(H_{inner} H_{c\_right} + B_{\text{COMB2}}) \quad (15)$$

As in the Compositional Matrix-Space model, the primary mode of semantic composition is matrix multiplication. Thus, the composition of two distributed representations cannot be reduced to element-wise addition or weighted average. However, unlike the Compositional Matrix-Space model, the proposed model does not multiply two word representations. Instead, there is an independent COMB layer,

into which the child node representations are sequentially fed. Since the composition process does not rely on pure matrix multiplication, it is no longer associative. For this reason, the Lifted Matrix-Space model better conforms to the principle of compositionality than the Compositional Matrix-Space model.

The proposed model differs from the Matrix-Vector RNN in that the word representations themselves are not part of the trained parameters. As noted earlier, training matrix representations for every word significantly increases computational complexity. In order to avoid such an issue, the proposed model imports a third-order tensor, W_{LIFT}, which transforms each pre-trained word embedding into a matrix. Although having a third-order tensor as a trained parameter is computationally expensive, it is generally more efficient than directly assigning a matrix parameter to every word.

Compared to the RNTN, our model transmits a larger number of activations across layers when given the same parameter count. In both models, the size of the third-order tensor is the dominant factor in determining the number of model parameters. The parameter count our model is approximately proportional to the number of activations (d), but the parameter count of the RNTN is approximately proportional to the cube of the number of activations (d^3). Therefore, our model can transmit the same number of activations with less model parameters.[2]

3.2 The Lifted Matrix-Space model augmented with TreeLSTM components

We augmented our model with TreeLSTM components to circumvent the problem of long-term dependencies (e.g., the vanishing gradient problem). Just as in TreeLSTM, the input/output gates and forget gates are functions of the child nodes' hidden states (18). But since the hidden states are represented as matrices, we reshape an $\sqrt{d} \times \sqrt{d}$ matrix into an $d \times 1$ column vector (17). The composition of child nodes proceeds as in the base model (21), and the cell state is computed accordingly (22). Lastly, the LSTM output is calculated and is reshaped back to an $\sqrt{d} \times \sqrt{d}$ matrix (23).

$$\vec{h}_{c\_left} = \text{VECTORIZE}(H_{c\_left}) \tag{16}$$

$$\vec{h}_{c\_right} = \text{VECTORIZE}(H_{c\_right}) \tag{17}$$

$$\begin{bmatrix} \vec{i} \\ \vec{f}_l \\ \vec{f}_r \\ \vec{o} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} \vec{h}_{c\_left} \\ \vec{h}_{c\_right} \end{bmatrix} + B \right) \tag{18}$$

$$H_{inner} = tanh(W_{\text{COMB}} H_{c\_left} + B_{\text{COMB1}}) \tag{19}$$

$$H_{candidate} = tanh(H_{inner} H_{c\_right} + B_{\text{COMB2}}) \tag{20}$$

$$\vec{g} = \text{VECTORIZE}(H_{candidate}) \tag{21}$$

$$\vec{c} = \vec{f}_l \odot \vec{c}_{left} + \vec{f}_r \odot \vec{c}_{right} + \vec{i} \odot \vec{g} \tag{22}$$

$$H = \text{TO-MATRIX}(\vec{o} \odot \vec{c}) \tag{23}$$

3.3 Simplified Lifted Matrix-Space model augmented with TreeLSTM components

We experimented on a simpler composition function that could potentially reduce the computational complexity. More specifically, we replaced the composition function in (21) with the following equation:

$$H = tanh(W_{\text{COMB}} H_{c\_left} H_{c\_right} + B_{\text{COMB}}) \tag{24}$$

$$\vec{g} = \text{VECTORIZE}(H_{candidate}) \tag{25}$$

The LIFT layer and the augmentation with TreeLSTM components remain unaltered. Compared to the model introduced in the previous subsection, we did not observe a significant reduction in the number of model parameters. However, reducing the number of composition steps can affect the overall training time and would be preferred.

Table 1 presents an overview of the models discussed so far and compares our model to the predecessors.

4 NLI experiments

Since the focus of this paper is the composition function, we compared the performance of our model to TreeLSTM.[3] After going through a good amount of tuning, the models without LSTM did not beat 70% accuracy and were left out.

4.1 Implementation details

We used the SPINN-PI-NT model architecture (Bowman et al. 2016) for the implementation of the TreeLSTM. The SPINN-PI-NT implements a standard TreeLSTM using stack and buffer data structures borrowed from parsing, rather than tree structures, to allow for efficient batching. We implemented our model by replacing the SPINN-PI-NT's composition function with ours and adding the LIFT layer.

Doing so is advantageous in two aspects. First, we can greatly reduce the time to train the models. Second, the models would be completely identical except for the LIFT layer and the composition function. Therefore, the two models are a 'minimal pair', and we can directly compare the performance of the composition functions.

We imported the 300D reference GloVe vectors (840B tokens; Pennington, Socher, and Manning 2014) for distributed representation of words. The word vectors are fed into the TreeLSTM and our model. This yields a vector for the TreeLSTM, and a matrix for our model. The output matrix of our model is transformed back into a vector.

We used the sentence pair model (Bowman et al. 2016) to classify a pair of sentences. This model encodes the premise sentence and the hypothesis sentence separately, and builds a feature vector from the outputs of the two encoders. In our case, the feature vector $\vec{x}_{classifier}$ consists of the vector representations of the premise sentence and the hypothesis

[2] Precisely speaking, the RNTN would be more efficient if $d^3 < d \times d_{emb}$. However, it would follow that $d = \sqrt{d_{emb}}$, and this means that the size of the hidden states is significantly smaller than that of the word embeddings. Then the model would lose a bulk of information in the word embeddings from the offset.

[3] While our model is intended to replace extant composition functions for better performance and scalability, it is possible that different composition functions are appropriate for deeper neural network models such as Irsoy and Cardie (2014). We leave it for future study to verify whether this is actually the case.

Model	Params.	S Train	S Test	M Train	M Test
Prior work					
CBOW (Williams, Nangia, and Bowman 2017)	–	–	80.6	–	65.2
BiLSTM (Williams, Nangia, and Bowman 2017)	2.8m	–	81.5	–	67.5
300D TreeLSTM (Bowman et al. 2016)	3.4m	84.4	80.9	–	–
300D SPINN-PI (Bowman et al. 2016)	3.7m	89.2	83.2	–	–
300D Neural Semantic Encoders (Munkhdalai and Yu 2016)	3.0m	86.2	84.6	–	–
Shortcut-Stacked (Nie and Bansal 2017)	–	–	**86.1**	–	**74.6**
This work					
300D TreeLSTM	3.4m	90.0	83.3	79.1	69.0
18×18D Lifted Matrix-Space	3.4m	91.8	84.2	76.5	**69.6**
20×20D Lifted Matrix-Space	4.2m	89.5	**84.4**	–	–
18×18D Simplified Lifted Matrix-Space	3.4m	91.0	83.6	–	–

Table 2: Results on SNLI and MultiNLI inference classification. *Params.* is the approximate number of model parameters, *S Train*, and *S Test* are the classification accuracies (%) on the entire SNLI training set and test set, respectively. *M Train*, and *M Test* are the classification accuracies (%) on the MultiNLI matched training set and test set, respectively.

sentence, the difference between the two, and their element-wise product.

$$\vec{x}_{classifier} = \begin{bmatrix} \vec{h}_{premise} \\ \vec{h}_{hypothesis} \\ \vec{h}_{premise} - \vec{h}_{hypothesis} \\ \vec{h}_{premise} \odot \vec{h}_{hypothesis} \end{bmatrix} \quad (26)$$

The feature vector is fed into an MLP that consists of two ReLU neural network layers and a softmax layer. In both models, the objective function is a sum of a cross-entropy loss function $\mathcal{L}_s$ and an L2 regularization term. Both models use stochastic gradient descent with the Adam optimizer (Kingma and Ba 2014). Dropout (Srivastava et al. 2014) is applied to the classifier and to the word embeddings. The MLP layer also utilizes batch normalization (Ioffe and Szegedy 2015).[4]

4.2 Test results and analysis

The models were first trained and tested on the *Stanford Natural Language Inference (SNLI) corpus* (Bowman et al. 2015). The SNLI corpus contains 570,152 pairs of natural language sentences that are labeled for *entailment*, *contradiction*, and *neutral*. It consists of sentences that were written and validated by humans. Along with the MultiNLI corpus introduced below, it is two orders of magnitude larger than other human-written resources for NLI tasks. The following example illustrates the general format of the corpus.

(27) **PREMISE:** A soccer game with multiple males playing.
HYPOTHESIS: Some men are playing a sport.
LABEL: Entailment

The models were additionally tested on the *Multi-Genre Natural Language Inference (MultiNLI) corpus* (Williams, Nangia, and Bowman 2017). The corpus consists of 433k pairs of examples, and each pair is labeled for *entailment*, *contradiction*, and *neutral*. The MultiNLI corpus has the same format as the SNLI corpus, so it is possible to train on both datasets at the same time. Two notable features distinguish the MultiNLI corpus from the SNLI corpus. First, it is collected from ten distinct genres of spoken and written English. This makes the dataset more representative of human language use. Second, the examples in the MultiNLI corpus are considerably longer than the ones in the SNLI corpus. The pair of sentences in (28) is an illustrative example. These sentences were extracted from a telephone speech, so they well represent how people speak.

(28) **GENRE:** Telephone speech
PREMISE: Yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or
HYPOTHESIS: August is a black out month for vacations in the company.
LABEL: Contradiction

The MultiNLI corpus can be divided into two subsets. The *matched* set is intended for those who want to test the quality of sentence representations derived from their encoders; its training set and the test set both contain texts from all ten genres. On the other hand, the training set of the *mismatched* set contains texts from only five genres, whereas the test set includes texts from all ten genres. We tested on the *matched* set because we want to reason whether the proposed models derive reasonable sentence representations.

Table 2 summarizes the results on SNLI and MultiNLI inference classification. As a baseline, we selected the TreeLSTM model with 300D activations. As for the SNLI dataset, we compared the TreeLSTM to two Lifted Matrix-Space models augmented with TreeLSTM components that differ in the size of activations; one uses an 18×18D matrix and has roughly the same parameter count as the baseline model.

[4]The source code and the trained parameters for all models are available at https://github.com/woojinchung/lms.

Model	Neg Dev (%)
300D TreeLSTM	68.8
18×18D Lifted Matrix-Space	78.1
20×20D Lifted Matrix-Space	81.3

Table 3: Results on SNLI inference classification. *Neg Dev* is the classification accuracy on the negation data in the development set.

The other uses a 20×20D matrix. It required a higher number of model parameters than the baseline model but showed the best performance. We additionally tested the simplified Lifted Matrix-Space model with 18×18D activations, which reduces the computational complexity.

The results show that the Lifted Matrix-Space models have an additional 0.9-1.1% gain over the TreeLSTM on the entire SNLI dataset. On the other hand, the simplified version only performs slightly better than the TreeLSTM, beating the baseline by 0.3%. This justified our decision to opt for a more complex composition function.

As for the evaluation on the MultiNLI dataset, we trained the 300D TreeLSTM and the 18×18D Lifted Matrix-Space model on both the MultiNLI and SNLI datasets. For both models, the same hyperparameter settings that gave the best results on the SNLI dataset were used. Again, the 18×18D Lifted Matrix-Space model outperformed the 300D TreeLSTM, beating the competitor by 0.6% on the test set.

In addition to testing the trained model on the entire SNLI and MultiNLI dataset, we extracted examples from the SNLI development set that contain a negation and evaluated our model on the extracted data. Specifically, we picked out sentence pairs that contain *not* or *n't* either in the premise or the hypothesis. For example, the two sentences in (29) contradict each other as the premise implies that there is water around the man but the hypothesis explicitly denies the existence with a negation.

(29) **PREMISE:** A man is kayaking in rough waters.
HYPOTHESIS: There isn't any water around the man.
LABEL: Contradiction

Understanding the semantics of negation has become a standard case study in NLI tasks (Icard III and Moss 2013). We expect our model to outperform the TreeLSTM on this task due to the more powerful composition function.

Table 3 summarizes the results on the negation data. Both of our models outperformed the TreeLSTM on the negation data. We speculate that the performance gain on the negation data enhanced the overall performance of our model.

4.3 Functional versus lexical meanings in LIFTed matrix representations

We compared the matrix representations of quantifiers and nominals to see whether the two categories show distinct patterns. The purpose of the inspection is to check whether the LIFT layer learns to represent quantifiers differently from content words. We hypothesize that the two categories show different patterns because the semantics of the former is more or less functional whereas the semantics of the latter is lexical (i.e., has the semantics of a content word).

Nominals were selected as a representative of content words because they are semantically simpler than other content words such as predicates. Eight nominals (*man*, *woman*, *dog*, *horse*, *boy*, *girl*, *car* , *child*) were arbitrarily chosen from the SNLI corpus and were compared against the selected quantifiers (*every*, *all*, *each*, *few*, *some*, *many*, *most*, *no*). We fed each 300D GloVe word embedding into the LIFT layer of the 400D Lifted Matrix-Space model trained on the SNLI corpus, and inspected the resulting matrix representations.

While the information seemed to be more scattered in the matrix representations of quantifiers, metrics over the matrix representations of the selected words did not show any clear trend. A more elaborate study is required for analyzing the interpretive properties of the LIFTed matrices.

5 Conclusion

In this paper, we have propose a model for semantic composition that utilizes matrix multiplication. The model converts vector representations of words into matrices, and composition of meanings reduces to matrix multiplication. Experimental results show that our model substantially outperformed the TreeLSTM on the SNLI corpus, especially regarding operator semantics. The model also obtains better performance on the MultiNLI corpus. While the model does not obtain the state-of-the-art performance, it is advantageous in that it can replace the composition layer of extant models without altering the general model design. By doing so, we expect that many tree structure-based models would benefit from better comprehension of operator semantics such as negation.

References

Barker, C., and Shan, C.-c. 2014. *Continuations and Natural Language*. Oxford University Press.

Baroni, M., and Zamparelli, R. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing* (October):1183–1193.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP 2015*, 632–642.

Bowman, S. R.; Gauthier, J.; Rastogi, A.; Gupta, R.; Manning, C. D.; and Potts, C. 2016. A Fast Unified Model for Parsing and Sentence Understanding. In *ACL*, 1466–1477.

Chen, D.; Socher, R.; Manning, C. D.; and Ng, A. Y. 2013. Learning New Facts From Knowledge Bases With Neural Tensor Networks and Semantic Word Vectors. In *Proc. of workshop at ICLR*, 1–4.

Chierchia, G., and McConnell-Ginet, S. 1990. *Meaning and grammar: An introduction to semantics*. Cambridge, MA: MIT Press.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural Language Processing (Almost) from Scratch. *JMLR* 12:2493–2537.

Dowty, D. 2007. Compositionality as an empirical problem. In *Direct Compositionality*. Oxford University Press.

Frege, G. 1892. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik* (1):25–50.

Hill, F.; Cho, K.; Jean, S.; and Bengio, Y. 2017. The representational geometry of word meanings acquired by neural machine translation models. *Machine Translation*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1–32.

Hochreiter, S.; Bengio, Y.; Frasconi, P.; and Schmidhuber, J. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Networks* 237–243.

Icard III, T. F., and Moss, L. S. 2013. Recent progress on monotonicity. *LiLT* 9(7):167–194.

Ioffe, S., and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*.

Irsoy, O., and Cardie, C. 2014. Deep recursive neural networks for compositionality in language. In *Proc. of NIPS*, volume 3, 2096–2104.

Kingma, D., and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *ICLR* 1–13.

Le, P., and Zuidema, W. 2015. Compositional Distributional Semantics with Long Short Term Memory. *arXiv:1503.02510*.

Mikolov, T.; Corrado, G.; Chen, K.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)* 1–12.

Mitchell, J., and Lapata, M. 2010. Composition in distributional models of semantics. *Cognitive science* 34(8):1388–1429.

Munkhdalai, T., and Yu, H. 2016. Neural Semantic Encoders. *arXiv:1607.04315*.

Nie, Y., and Bansal, M. 2017. Shortcut-Stacked Sentence Encoders for Multi-Domain Inference. In *Proceedings of RepEval 2017: The Second Workshop on Evaluating Vector Space Representations for NLP (to appear)*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* 1532–1543.

Rudolph, S., and Giesbrecht, E. 2010. Compositional Matrix-Space Models of Language. In *ACL*, 907–916.

Socher, R.; Lin, C. C.; Manning, C. D.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 129–136.

Socher, R.; Huval, B.; Manning, C. D.; and Ng, A. Y. 2012. Semantic Compositionality through Recursive Matrix-Vector Spaces. *Proc. of EMNLP 2012* 1201–1211.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive Deep Models for Semantic Compositionality over a Sentiment treebank. In *Proc. of EMNLP 2013*, 1631–1642.

Socher, R.; Manning, C. D.; and Ng, A. Y. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop* 1–9.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15:1929–1958.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proc. of ACL*, 1556–1566.

Williams, A.; Nangia, N.; and Bowman, S. R. 2017. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *arXiv:1704.05426*.

Yessenalina, A., and Cardie, C. 2011. Compositional Matrix-Space Models for Sentiment Analysis. *CL* 172–182.

Zanzotto, F. M.; Korkontzelos, I.; Fallucchi, F.; and Manandhar, S. 2010. Estimating linear models for compositional distributional semantics. In *Proc. of COLING 2016*, volume 1, 1263–1271.

Zhu, X.; Sobhani, P.; and Guo, H. 2015. Long Short-Term Memory Over Tree Structures. In *Proc. of ICML*, 1604–1612.